Dart language cheatsheet

Dart's commonly used features

Syntax	Usage	Note
Using literals	‘Substitution ${val}’	Puts the value of val into a string literal. Equivalent: ‘Substitution’ + val
	<type>[ ]	Creates an object of type List<type>.
	const [1, 2, 3]	Creates a compile-time constant list.
	= { }	Initializes a map. Equivalent: new Map<>();
Declaring fields	var	Generic var with type inference
	final	Same as var but cannot be reassigned
	const	Compile-time constant
Checking types	as	Typecast
	is	instanceof
	is!	!instanceof

Chaining method calls	`a..b = true..c = 5;`	Cascade used for chaining access to methods and other members. Equivalent: `a.b = true; a.c = 5;`
Dealing with null	`b ??= val;`	If `b` is null, assign the value of `val` to `b`; otherwise, `b` stays the same.
	`a = value ?? 0;`	If value is null, set `a` to 0. Otherwise, set `a` to value.
	`a?.b`	Conditional access. Equivalent: `a == null ? null : a.b`
Implementing functions	`fn({bool bold = false, bool hidden = false})`	Named params with default values.
	`int incr(int a) => a + 1;`	Single return statement can be abbreviated.

Handling exceptions	`try {...}`
	`on MyException {...}`
	`catch (e) {...}`
	`finally {...}`
		Use `on` to catch a type. Use `catch` to catch an instance.

Implementing constructors	`Point(this.x, this.y);`	Normal constructor
	`factory Point(int x, int y) => ...;`	Factory constructor Use `factory` when implementing a constructor that doesn't always create a new instance.

	`Point.fromJson(Map json) {`
	`x = json['x'];`
	`y = json['y'];`
	`}`
		Named constructor

	`Point.alongXAxis(num x) : this(x, 0);`	Delegating constructor
	`const ImmutablePoint(this.x, this.y);`	Const constructor

	Produces an object that will never change. All fields have to be final.
`Point.fromJson(Map jsonMap) : x = jsonMap['x'], y = jsonMap['y'];`	Initializer list Initializer lists are handy when setting up final fields.

By Javi 🚀 🏝 source: https://dart.dev/guides/language/cheatsheet - 2021